

# **Operating Systems - Assignment**

**COMP2006**

**Jordan Yeo**  
**17727626**

## Contents

<b>1</b>	<b>Process MSSV</b>	<b>1</b>
<b>2</b>	<b>Thread MSSV</b>	<b>1</b>
<b>3</b>	<b>Testing</b>	<b>3</b>
3.1	Errors . . . . .	3
3.2	Input Files . . . . .	3
3.3	Expected Results . . . . .	4
3.4	Actual Results . . . . .	5
<b>4</b>	<b>README</b>	<b>6</b>
4.1	Purpose . . . . .	6
4.2	Running the Program . . . . .	6
4.3	Files . . . . .	6
	<b>References</b>	<b>7</b>
<b>5</b>	<b>Appendices</b>	<b>8</b>

## Introduction

The following report is for Operating Systems Assignment for 2017. It will detail how mutual exclusion was achieved for processes and threads. Also the processes and threads that accessed shared resources. Submitted alongside this report is a README, source code and test inputs and outputs.

## 1 Process MSSV

For this implementation of the solution, mutual exclusion is achieved by having the parent (consumer) wait for all child (producer) processes to complete execution before continuing. This waiting is implemented by having the parent wait for the `semParent` semaphore before acquiring mutex lock to check if all children have finished. Once a child completes execution it waits for the `semParent` semaphore before acquiring a mutex lock for itself to update a shared resource, indicating it is finished.

```
wait(semParent)
wait(semMutex)
// critical section
signal(semMutex)
signal(semParent)
```

To ensure mutual exclusion for the shared resources (`buffer1`, `buffer2` and `counter`) the child waited to acquire a mutex lock before entering its critical section to modify the shared resources. Since `buffer1` was never modified and only read from this did not require a mutex lock to be obtained before reading.

The semaphore required (`semMutex` and `semParent`), `buffer1`, `buffer2` and `counter` were implemented using shared memory. The following POSIX shared memory functions were used to create shared memory, and the corresponding functions to close the shared memory:

```
shm_link()
ftruncate()
mmap()
shm_unlink()
close()
munmap()
```

Zombie processes were killed with the use of `signal(SIGCHLD, SIG_IGN)`.

## 2 Thread MSSV

To achieve mutual exclusion in the multi-threaded program, the parent (consumer) must wait for all threads (producer) to complete execution. The parent uses `pthread_lock_mutex()` to lock the mutex then performs a `pthread_cond_wait()` on a condition variable. The condition variable represents how many threads are still executing. Once the thread completes execution it acquires a mutex lock to alter the condition variable.

In the thread MSSV the shared resources are declared as global variables. Threads have access to the variables declared global in the parent. Before a thread could access the shared resources it

would first need to acquire a mutex lock. The function *pthread\_lock\_mutex()* blocks the caller if the mutex is in use by another. It can then alter the shared resources, counter and buffer2, before releasing the mutex lock.

Threads

### 3 Testing

#### 3.1 Errors

There are no known errors in the process MSSV and the thread MSSV. Care has been taken to ensure potential memory leaks are mitigated, by using the appropriate measures to free allocated memory. Memory leaks are not present in the testing of each MSSV currently performed.

#### 3.2 Input Files

	6	2	4	5	3	9	1	8	7
	5	1	9	7	2	8	6	3	4
	8	3	7	6	1	4	2	9	5
	1	4	3	8	6	5	7	2	9
<i>specTest</i> =	9	5	8	2	4	7	3	6	1
	7	6	2	3	9	1	4	5	8
	3	7	1	9	5	6	8	4	2
	4	9	6	1	8	2	5	7	3
	2	8	5	4	7	3	9	1	6
	2	2	4	5	3	9	1	8	7
	5	1	9	7	2	8	6	3	4
	8	3	7	6	1	4	2	9	5
	1	4	3	8	6	5	7	2	9
<i>testFail</i> =	9	5	8	2	4	7	3	6	1
	7	6	2	3	9	1	4	5	8
	3	7	1	9	5	6	8	4	2
	4	9	6	1	8	2	5	7	3
	2	8	5	4	7	3	9	1	6
	2	2	4	5	3	9	1	8	7
	5	1	9	7	2	8	6	3	4
	8	3	7	6	1	4	2	9	5
	1	4	3	8	6	5	7	2	9
<i>multiFail</i> =	9	5	8	2	4	7	3	6	3
	7	6	2	3	9	1	4	5	8
	3	7	1	9	5	6	8	4	2
	4	9	6	1	8	2	5	7	3
	2	8	5	4	7	3	9	1	6

### 3.3 Expected Results

*specTest*

```
row 1 is valid
row 2 is valid
row 3 is valid
row 4 is valid
row 5 is valid
row 6 is valid
row 7 is valid
row 8 is valid
row 9 is valid
9 out of 9 columns are valid
9 out of 9 sub-grids are valid

There are 27 valid sub-grids, and thus the solution is valid
```

*testFail*

```
row 1 is invalid
row 2 is valid
row 3 is valid
row 4 is valid
row 5 is valid
row 6 is valid
row 7 is valid
row 8 is valid
row 9 is valid
8 out of 9 columns are valid
8 out of 9 sub-grids are valid

There are 24 valid sub-grids, and thus the solution is invalid
```

*multiFail*

```
row 1 is invalid
row 2 is valid
row 3 is valid
row 4 is valid
row 5 is invalid
row 6 is valid
row 7 is valid
row 8 is valid
row 9 is valid
7 out of 9 columns are valid
7 out of 9 sub-grids are valid

There are 21 valid sub-grids, and thus the solution is invalid
```

### 3.4 Actual Results

## 4 README

### 4.1 Purpose

The program validates an input file containing a sudoku solution. There are two versions. One utilising processes and the other utilising threads.

### 4.2 Running the Program

To compile the C files into an executable format.

```
make
```

To run the program

```
make run
```

This will only let you run with the preset parameters and they will need to be altered in the Makefile to test other *input files* and *delays*

```
INPUT: ../testFiles/specTest.txt  
DELAY: 10
```

Between each time the program is run, the following command should be entered and executed. This is to delete the logfile produced from an invalid test file.

```
make clean
```

### 4.3 Files

partA

- Makefile
- mssv.c
- mssv.h

partB

- Makefile
- mssv.c
- mssv.h

testFiles

- test files

## References

- [1] *Interprocess communication using POSIX Shared Memory in Linux*. URL: <https://www.softprayog.in/programming/interprocess-communication-using-posix-shared-memory-in-linux>.
- [2] *POSIX Semaphores*. URL: <https://www.softprayog.in/programming/posix-semaphores>.
- [3] *POSIX Threads Programming in C*. URL: <https://www.softprayog.in/programming/posix-threads-programming-in-c>.
- [4] *POSIX Threads Synchronization in C*. URL: <https://www.softprayog.in/programming/posix-threads-synchronization-in-c>.



## 5 Appendices